

4. Context-free Languages, Context-free Grammars, and Pushdown Automata

Contents

- Context-Free Grammars
- Parse-trees
- Pushdown Automata
- PDA and CFL
- Pumping Lemma for CFL

Context-Free Grammars

- All regular languages, even those containing an infinite number of words, can be defined using REs with a finite number of terms.
- Some non-regular languages can also be finitely defined, but not using REs.
- Here, we will consider Context-Free Languages (CFL), which contain all regular languages and many non-regular ones, and which can be defined Context-Free Grammars (CFG).

Context-Free Grammars

- All regular languages, even those containing an infinite number of words, can be defined using REs with a finite number of terms.
- Some non-regular languages can also be finitely defined, but not using REs.
- Here, we will consider Context-Free Languages (CFL), which contain all regular languages and many non-regular ones, and which can be defined Context-Free Grammars (CFG).

Context-Free Grammars

- All regular languages, even those containing an infinite number of words, can be defined using REs with a finite number of terms.
- Some non-regular languages can also be finitely defined, but not using REs.
- Here, we will consider Context-Free Languages (CFL), which contain all regular languages and many non-regular ones, and which can be defined Context-Free Grammars (CFG).

Context-Free Grammars

- A CFG consists of a finite sequence of simple **production** (or **rewrite**) **rules**, each of which has a **single non-terminal** symbol on the left-hand side and an expression on the right.

- **Example**

$$S \rightarrow AB \mid B$$
$$A \rightarrow B \mid CC$$
$$B \rightarrow b$$
$$C \rightarrow c$$

Context-Free Grammars

Example

$$S \rightarrow AB \mid B$$
$$A \rightarrow B \mid CC$$
$$B \rightarrow b$$
$$C \rightarrow c$$

- The ‘|’ character indicates an alternative (‘or’).
- Symbols in upper case conventionally denote **non-terminals**, which occur only once on the LHS (they can of course occur more than once on the RHS) and denote **sub-languages**.
- Lower case symbols denote **terminals**, which are the elements of the language’s **alphabet**.

Context-Free Grammars

Example

$$S \rightarrow AB \mid B$$
$$A \rightarrow B \mid CC$$
$$B \rightarrow b$$
$$C \rightarrow c$$

- The ‘|’ character indicates an alternative (‘or’).
- Symbols in upper case conventionally denote **non-terminals**, which occur only once on the LHS (they can of course occur more than once on the RHS) and denote **sub-languages**.
- Lower case symbols denote **terminals**, which are the elements of the language’s **alphabet**.

Context-Free Grammars

Example

$$S \rightarrow AB \mid B$$
$$A \rightarrow B \mid CC$$
$$B \rightarrow b$$
$$C \rightarrow c$$

- The ‘|’ character indicates an alternative (‘or’).
- Symbols in upper case conventionally denote **non-terminals**, which occur only once on the LHS (they can of course occur more than once on the RHS) and denote **sub-languages**.
- Lower case symbols denote **terminals**, which are the elements of the language’s **alphabet**.

Context-Free Grammars

Example

$$S \rightarrow AB \mid B$$
$$A \rightarrow B \mid CC$$
$$B \rightarrow b$$
$$C \rightarrow c$$

- This CFG produces the words:

- $bb \quad S \rightarrow AB \rightarrow BB \rightarrow bb$

- ccb

- b

Context-Free Grammars

Example

$$S \rightarrow AB \mid B$$
$$A \rightarrow B \mid CC$$
$$B \rightarrow b$$
$$C \rightarrow c$$

- This CFG produces the words:
 - $bb \quad S \rightarrow AB \rightarrow BB \rightarrow bb$
 - $ccb \quad S \rightarrow AB \rightarrow CCB \rightarrow ccb$
 - b

Context-Free Grammars

Example

$$S \rightarrow AB \mid B$$
$$A \rightarrow B \mid CC$$
$$B \rightarrow b$$
$$C \rightarrow c$$

- This CFG produces the words:
 - $bb \quad S \rightarrow AB \rightarrow BB \rightarrow bb$
 - $ccb \quad S \rightarrow AB \rightarrow CCB \rightarrow ccb$
 - $b \quad S \rightarrow B \rightarrow b$

Definition

- A **context-free grammar** (CFG) is a 4-tuple (N, Σ, R, S) , where:
 - N is a set of **non-terminal** symbols – also called **variables**
 - Σ is a set of symbols such that $\Sigma \cap N = \emptyset$, called **terminals**
 - R is a set of **production rules**, where each rule is pair, one element being a non-terminal, the other a string of non-terminals and terminals
 - $S \in N$ is the start symbol (variable)
- They are called **context-free** since the production rules can be applied regardless of the ‘context’ of a non-terminal, i.e., the symbols that surround it.

Definition

- A **context-free grammar** (CFG) is a 4-tuple (N, Σ, R, S) , where:
 - N is a set of **non-terminal** symbols – also called **variables**
 - Σ is a set of symbols such that $\Sigma \cap N = \emptyset$, called **terminals**
 - R is a set of **production rules**, where each rule is pair, one element being a non-terminal, the other a string of non-terminals and terminals
 - $S \in N$ is the start symbol (variable)
- They are called **context-free** since the production rules can be applied regardless of the ‘context’ of a non-terminal, i.e., the symbols that surround it.

Definition

- A **context-free grammar** (CFG) is a 4-tuple (N, Σ, R, S) , where:
 - N is a set of **non-terminal** symbols – also called **variables**
 - Σ is a set of symbols such that $\Sigma \cap N = \emptyset$, called **terminals**
 - R is a set of **production rules**, where each rule is pair, one element being a non-terminal, the other a string of non-terminals and terminals
 - $S \in N$ is the start symbol (variable)
- They are called **context-free** since the production rules can be applied regardless of the ‘context’ of a non-terminal, i.e., the symbols that surround it.

Definition

- A **context-free grammar** (CFG) is a 4-tuple (N, Σ, R, S) , where:
 - N is a set of **non-terminal** symbols – also called **variables**
 - Σ is a set of symbols such that $\Sigma \cap N = \emptyset$, called **terminals**
 - R is a set of **production rules**, where each rule is pair, one element being a non-terminal, the other a string of non-terminals and terminals
 - $S \in N$ is the start symbol (variable)
- They are called **context-free** since the production rules can be applied regardless of the ‘context’ of a non-terminal, i.e., the symbols that surround it.

Definition

- A **context-free grammar** (CFG) is a 4-tuple (N, Σ, R, S) , where:
 - N is a set of **non-terminal** symbols – also called **variables**
 - Σ is a set of symbols such that $\Sigma \cap N = \emptyset$, called **terminals**
 - R is a set of **production rules**, where each rule is pair, one element being a non-terminal, the other a string of non-terminals and terminals
 - $S \in N$ is the start symbol (variable)
- They are called **context-free** since the production rules can be applied regardless of the ‘context’ of a non-terminal, i.e., the symbols that surround it.

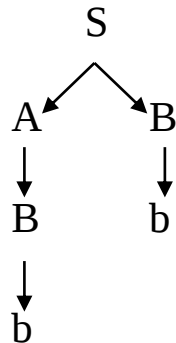
Definition

- A **context-free grammar** (CFG) is a 4-tuple (N, Σ, R, S) , where:
 - N is a set of **non-terminal** symbols – also called **variables**
 - Σ is a set of symbols such that $\Sigma \cap N = \emptyset$, called **terminals**
 - R is a set of **production rules**, where each rule is pair, one element being a non-terminal, the other a string of non-terminals and terminals
 - $S \in N$ is the start symbol (variable)
- They are called **context-free** since the production rules can be applied regardless of the ‘context’ of a non-terminal, i.e., the symbols that surround it.

$$\begin{aligned} S &\rightarrow AB \mid B \\ A &\rightarrow B \mid CC \\ B &\rightarrow b \\ C &\rightarrow c \end{aligned}$$

Parse Trees

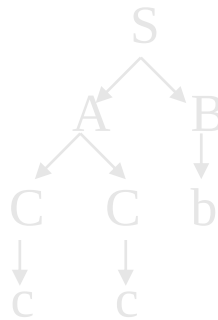
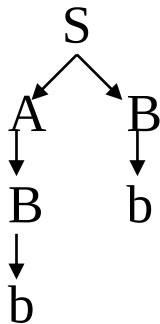
- Each word in the language represented by a CFG is generated by applying a production rule to the start symbol (conventionally 'S') and applying further rules to the non-terminals in the strings so produced, until **only terminals** are left.
- This construction generates a tree with the start symbol at the root and **a word** in the language at the leaves (leaves should be read left to right).



Parse Trees (example)

$$S \rightarrow AB \mid B$$
$$A \rightarrow B \mid CC$$
$$B \rightarrow b$$
$$C \rightarrow c$$

- Applied to the above grammar, this procedure generates all, and only, the following trees:

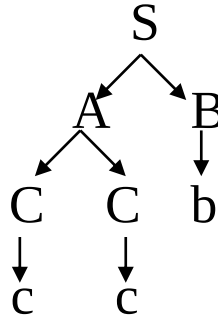
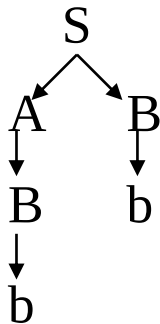


- This language therefore consists of the set of words {bb, ccb, b}. The trees show how each word is generated from the grammar. This process is called **parsing** and the trees are called **parse trees**.

Parse Trees (example)

$$S \rightarrow AB \mid B$$
$$A \rightarrow B \mid CC$$
$$B \rightarrow b$$
$$C \rightarrow c$$

- Applied to the above grammar, this procedure generates all, and only, the following trees:

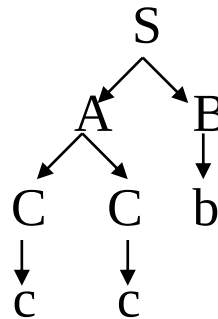
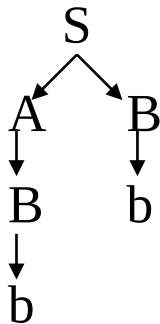


- This language therefore consists of the set of words {bb, ccb, b}. The trees show how each word is generated from the grammar. This process is called **parsing** and the trees are called **parse trees**.

Parse Trees (example)

$$S \rightarrow AB \mid B$$
$$A \rightarrow B \mid CC$$
$$B \rightarrow b$$
$$C \rightarrow c$$

- Applied to the above grammar, this procedure generates all, and only, the following trees:

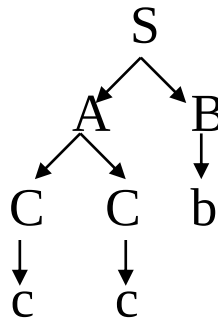
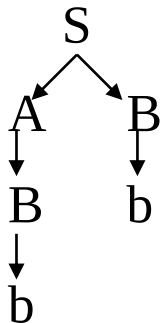


- This language therefore consists of the set of words {bb, ccb, b}. The trees show how each word is generated from the grammar. This process is called **parsing** and the trees are called **parse trees**.

Parse Trees (example)

$$S \rightarrow AB \mid B$$
$$A \rightarrow B \mid CC$$
$$B \rightarrow b$$
$$C \rightarrow c$$

- Applied to the above grammar, this procedure generates all, and only, the following trees:



- This language therefore consists of the set of words {bb, ccb, b}. The trees show how each word is generated from the grammar. This process is called **parsing** and the trees are called **parse trees**.

$a^n b^n$ revisited

- The language $\{bb, ccb, b\}$ is finite and, of course, regular. But the same notation may be used to define non-regular languages.
- The language $\{a^n b^n \mid n > 0\}$, is known to be non-regular and is given by the CFG:
$$\begin{aligned} S &\rightarrow AB \mid ASB \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$
- Note the **recursion** here, the non-terminal S appearing on both the left and right hand sides.
- Like a 'loop' in an FSA, recursion generates infinite languages.

$a^n b^n$ revisited

- The language $\{bb, ccb, b\}$ is finite and, of course, regular. But the same notation may be used to define non-regular languages.
- The language $\{a^n b^n \mid n > 0\}$, is known to be non-regular and is given by the CFG:
$$\begin{aligned} S &\rightarrow AB \mid ASB \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$
- Note the **recursion** here, the non-terminal S appearing on both the left and right hand sides.
- Like a 'loop' in an FSA, recursion generates infinite languages.

The Null Word

- The *production rules* also allow us to refer to the null word, denoted by the symbol ε .
- This symbol is not a member of the alphabet of any language, nor is it a valid non-terminal (so it may not appear on the left-hand side of any production rule).
- **Example:** The language $\{a^n b^n \mid n \geq 0\}$, which includes the null word, is defined by:

$$S \rightarrow \varepsilon \mid ASB$$

$$A \rightarrow a$$

$$B \rightarrow b$$

or shorter

$$S \rightarrow \varepsilon \mid aSb$$

Note: For convenience, we **do not** allow a non-terminal, say A , to be on the LHS of **only** one rule of the type $A \rightarrow \varepsilon$. (Or we can assume that A has been replaced by ε in all RHS it appears, and $A \rightarrow \varepsilon$ is deleted)

The Null Word

- The *production rules* also allow us to refer to the null word, denoted by the symbol ε .
- This symbol is not a member of the alphabet of any language, nor is it a valid non-terminal (so it may not appear on the left-hand side of any production rule).
- **Example:** The language $\{a^n b^n \mid n \geq 0\}$, which includes the null word, is defined by:

$$S \rightarrow \varepsilon \mid ASB$$

$$A \rightarrow a$$

$$B \rightarrow b$$

or shorter

$$S \rightarrow \varepsilon \mid aSb$$

Note: For convenience, we **do not** allow a non-terminal, say A , to be on the LHS of **only** one rule of the type $A \rightarrow \varepsilon$. (Or we can assume that A has been replaced by ε in all RHS it appears, and $A \rightarrow \varepsilon$ is deleted)

The Null Word

- The *production rules* also allow us to refer to the null word, denoted by the symbol ε .
- This symbol is not a member of the alphabet of any language, nor is it a valid non-terminal (so it may not appear on the left-hand side of any production rule).
- **Example:** The language $\{a^n b^n \mid n \geq 0\}$, which includes the null word, is defined by:

$$S \rightarrow \varepsilon \mid ASB$$

$$A \rightarrow a$$

$$B \rightarrow b$$

or shorter

$$S \rightarrow \varepsilon \mid aSb$$

Note: For convenience, we **do not** include non-terminals, say A , to be on the LHS if **only** the rule $A \rightarrow \varepsilon$ exists. (In that case, we can assume that A has always been replaced by ε in all RHS it appears, and $A \rightarrow \varepsilon$ is deleted and the rules with A in the RHS modified)

More examples

- For the language $\{a^n b^n \mid n \geq 0\} \cup \{b^n a^n \mid n \geq 0\}$, note that it is the union of two simpler languages:

$\{a^n b^n \mid n \geq 0\}$ with CFG $S_1 \rightarrow \varepsilon \mid aS_1 b$

and

$\{b^n a^n \mid n \geq 0\}$ with CFG $S_2 \rightarrow \varepsilon \mid bS_2 a$

So

$$S \rightarrow S_1 \mid S_2$$

$$S_1 \rightarrow \varepsilon \mid aS_1 b$$

$$S_2 \rightarrow \varepsilon \mid bS_2 a$$

More examples (cont.)

Palindrome language with $\Sigma = \{a, b\}$

e.g. aa, bab, aabaa, abba

More examples (cont.)

Palindrome language with $\Sigma = \{a, b\}$

e.g. aa, bab, aabaa, abba

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$$

More examples (cont.)

Addition language: $\{a^n b^m c^{n+m} \mid n \geq 0, m \geq 0\}$

More examples (cont.)

Addition language: $\{a^n b^m c^{n+m} \mid n \geq 0, m \geq 0\}$

$$S \rightarrow aSc \mid X$$

More examples (cont.)

Addition language: $\{a^n b^m c^{n+m} \mid n \geq 0, m \geq 0\}$

$$S \rightarrow aSc \mid X$$

$$X \rightarrow bXc \mid \varepsilon$$

More examples (cont.)

RE $(a \cup b)^*aba^*$

More examples (cont.)

$$\text{RE } (a \cup b)^* aba^*$$

$$S \rightarrow XabY$$

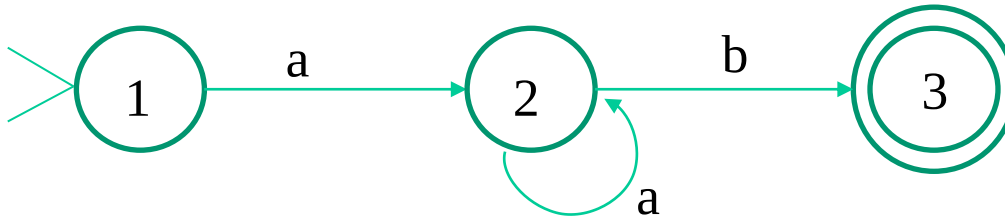
$$X \rightarrow aX \mid bX \mid \varepsilon$$

$$Y \rightarrow aY \mid \varepsilon$$

More examples (cont.)

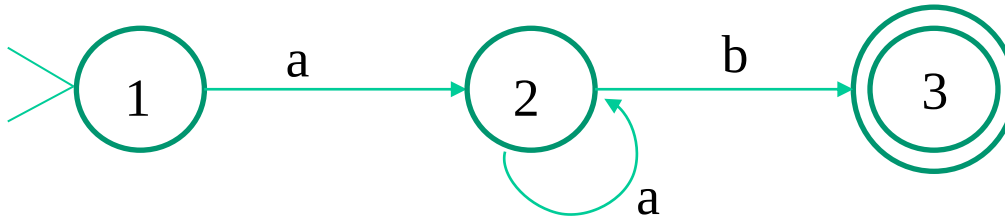
- For a regular language, one can construct a CFG easily from a DFA that recognizes the language (if the DFA is known/can be easily constructed) as follows:
- Make a non-terminal R_i for each state q_i of the DFA.
- Add to the CFG the rule $R_i \rightarrow aR_j$ if $\delta(q_i, a) = q_j$ is a transition of the DFA.
- Add the rule $R_i \rightarrow \varepsilon$ if q_i is an accept state of the DFA.
- Make R_0 the start variable of the CFG, where q_0 is the start state of the DFA.
- Verify on your own that the CFG indeed generates the same language that the DFA recognizes.

DFA to CFG example (cont.)



$$R_1 \rightarrow aR_2$$

DFA to CFG example (cont.)



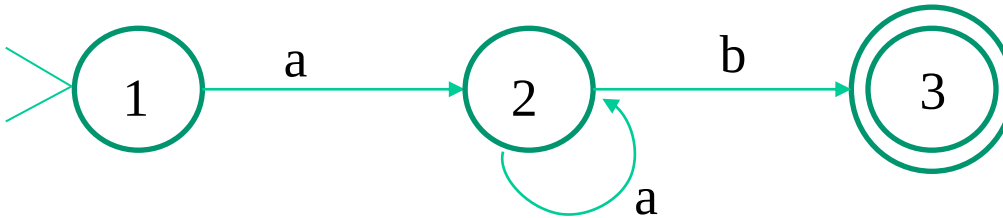
$$R_1 \rightarrow aR_2$$

$$R_2 \rightarrow aR_2$$

$$R_2 \rightarrow bR_3$$

$$R_3 \rightarrow \varepsilon$$

DFA to CFG example (cont.)



$$R_1 \rightarrow aR_2$$

$$R_2 \rightarrow aR_2$$

$$R_2 \rightarrow b$$

Pushdown Automata: Example

- Imagine an intuitive design for a machine that accepts the language $L = \{a^n b^n | n \geq 0\}$
- Suppose that this machine has a **stack** with an **infinite capacity**.
- When presented with a word at its input, it reads the symbols one at a time:
 - When it reads an 'a', it **pushes** some symbol, say '*', onto the stack.
 - When it reads a 'b', it **pops** a '*' off the stack.

Pushdown Automata: Example

- Imagine an intuitive design for a machine that accepts the language $L = \{a^n b^n | n \geq 0\}$
- Suppose that this machine has a **stack** with an **infinite capacity**.
- When presented with a word at its input, it reads the symbols one at a time:
 - When it reads an 'a', it **pushes** some symbol, say '*', onto the stack.
 - When it reads a 'b', it **pops** a '*' off the stack.

Pushdown Automata: Example

- If the stack is empty when no 'b's are left, then there must have been an equal number of 'a's and 'b's.
- This machine could accept all words of L and reject all words not in L.
- That is, it could recognise L, which we know to be **context free**.
- This machine belongs to class of **Pushdown Automata** (PDA).
- A PDA is a **state machine** plus a **stack**.

Pushdown Automata: Example

- If the stack is empty when no 'b's are left, then there must have been an equal number of 'a's and 'b's.
- This machine could accept all words of L and reject all words not in L.
- That is, it could recognise L, which we know to be **context free**.
- This machine belongs to class of **Pushdown Automata** (PDA).
- A PDA is a **state machine** plus a **stack**.

Definition

- A **pushdown automaton (PDA)** is given by a 6-tuple $(Q, \Sigma, \Gamma, \delta, s_0, F)$, where:
 - Q is a finite set of states
 - Σ is the input alphabet
 - Γ is the set of stack symbols
 - $s_0 \in Q$ is the initial state.
 - $\delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})) \times (Q \times (\Gamma \cup \{\varepsilon\})^*)$ is the transition relation.
 - $F \subseteq Q$ is the set of accept states.

Definition

- A **pushdown automaton (PDA)** is given by a 6-tuple $(Q, \Sigma, \Gamma, \delta, s_0, F)$, where:
 - Q is a finite set of states
 - Σ is the input alphabet
 - Γ is the set of stack symbols
 - $s_0 \in Q$ is the initial state.
 - $\delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})) \times (Q \times (\Gamma \cup \{\varepsilon\})^*)$ is the transition relation.
 - $F \subseteq Q$ is the set of accept states.

Definition

- A **pushdown automaton (PDA)** is given by a 6-tuple $(Q, \Sigma, \Gamma, \delta, s_0, F)$, where:
 - Q is a finite set of states
 - Σ is the input alphabet
 - Γ is the set of stack symbols
 - $s_0 \in Q$ is the initial state.
 - $\delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})) \times (Q \times (\Gamma \cup \{\varepsilon\})^*)$ is the transition relation.
 - $F \subseteq Q$ is the set of accept states.

Operation

- A PDA **accepts** string w if starting with an empty stack there is a computation **ending in an accept state with the stack again empty**. Otherwise, the string is **rejected**.
- As suggested above, the machine operates by beginning in the start state with an empty stack, and following transitions as specified in the transition relation.

Transitions

- Consider transition $((s, a, u), (t, v))$:
 - An automaton can follow this transition if it is in state s , 'a' is the next symbol on the input tape, and word 'u' is at the top of stack.
 - The automaton then moves to state t , popping 'u' from the stack and pushing 'v' onto the stack.
 - Note, 'a', 'u' and 'v' may all be ε , the null symbol.

Transitions

- Consider transition $((s, a, u), (t, v))$:
 - An automaton can follow this transition if it is in state s , ' a ' is the next symbol on the input tape, and word ' u ' is at the top of stack.
 - The automaton then moves to state t , popping ' u ' from the stack and pushing ' v ' onto the stack.
 - Note, ' a ', ' u ' and ' v ' may all be ε , the null symbol.

Transitions

- Consider transition $((s, a, u), (t, v))$:
 - An automaton can follow this transition if it is in state s , ' a ' is the next symbol on the input tape, and word ' u ' is at the top of stack.
 - The automaton then moves to state t , popping ' u ' from the stack and pushing ' v ' onto the stack.
 - Note, ' a ', ' u ' and ' v ' may all be ε , the null symbol.

Example

- The following PDA accepts the language $\{a^n b^n \mid n \geq 0\}$:

- $Q = \{s_0, s, f\}$

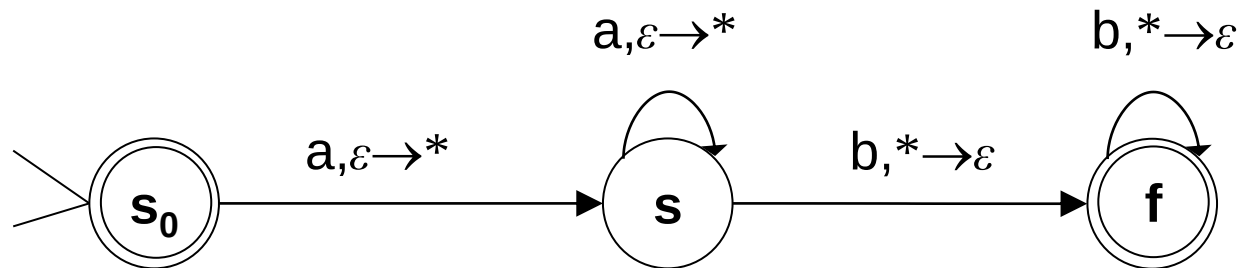
- $\Sigma = \{a, b\}$

- $\Gamma = \{*\}$

- $F = \{s_0, f\}$

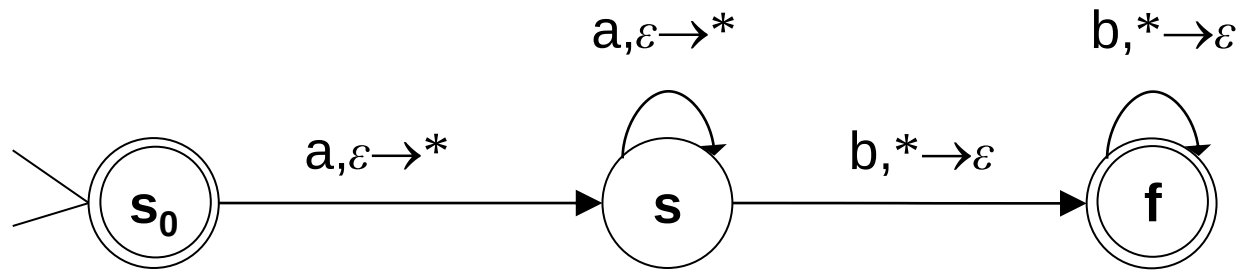
- δ is the set containing:

- $((s_0, a, \varepsilon), (s, *)), ((s, a, \varepsilon), (s, *)), ((s, b, *), (f, \varepsilon)), ((f, b, *), (f, \varepsilon))$



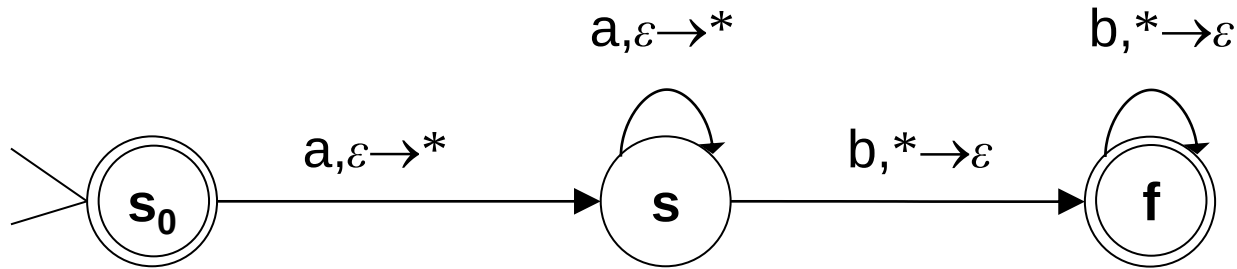
$x, y \rightarrow z$: transition when reading x , popping out y , and pushing in z

Example:



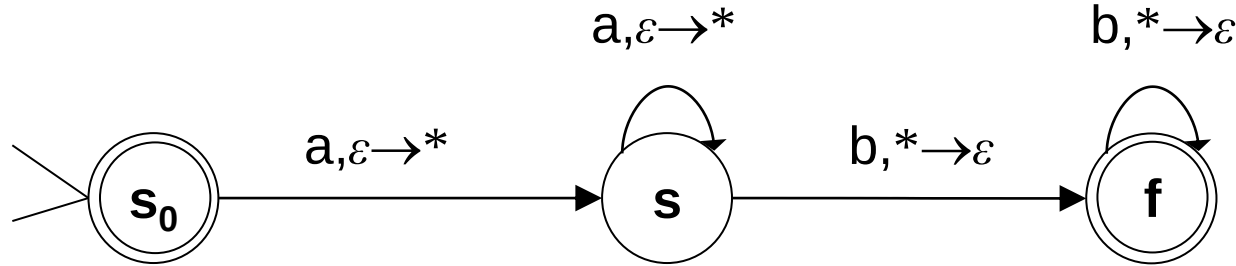
- Here, $a, \varepsilon \rightarrow *$ is read as ‘read character a from the input, pop ε from the stack (pop empty is don’t pop anything) and push $*$ onto the stack’.

Example



- The PDA accepts the empty word.
- For any other word it:
 - pushes a ‘*’ on the stack for each ‘a’ that is observed
 - allows one transition from ‘a’s to ‘b’s in the input string
 - pops one ‘*’ from the stack for each ‘b’ observed.
- If the stack is empty and the input has been read the number of ‘a’s and ‘b’s are balanced.

Example



- Consider input aabb:

State	Input	Stack
s_0	<u>a</u> abb	ε
s	a <u>a</u> bb	*
s	aa <u>b</u> b	**
f	aab <u>b</u>	*
f	aabb <u> </u>	ε

PDAs and CFGs

- In an earlier lecture it was shown that the class of regular languages corresponds exactly to the class of languages that could be accepted by FSA.
- That is, a FSA could be built to recognise any given regular language.
- A similar relationship holds between PDA and CFLs.
- The example above demonstrated one CFL that could be recognised by a PDA.

PDAs and CFGs

- In an earlier lecture it was shown that the class of regular languages corresponds exactly to the class of languages that could be accepted by FSA.
- That is, a FSA could be built to recognise any given regular language.
- A similar relationship holds between PDA and CFLs.
- The example above demonstrated one CFL that could be recognised by a PDA ($L = \{a^n b^n \mid n \geq 0\}$)

PDA for any given CFL

- **Theorem:** There exists an algorithm that, given any context-free grammar, constructs a PDA such that the language accepted by the PDA is exactly that described by the CFG.
- Proof:
 - Construct a PDA with two states. The first is the start state, s_0 , and the second, f , is an accept state.
 - The tape alphabet for the PDA is the same as that for the CFG.

PDA for any given CFL

- **Theorem:** There exists an algorithm that, given any context-free grammar, constructs a PDA such that the language accepted by the PDA is exactly that described by the CFG.
- **Proof:** We constructively define the PDA: $(Q, \Sigma, \Gamma, \delta, s_0, F)$
 - Construct a PDA with two states. The first is the start state, s_0 , and the second, f , is an accept state.
 - The tape alphabet Σ for the PDA is the same as that for the CFG.
 - The stack alphabet is all the symbols (both terminal and non-terminal) of the CFG.

PDA for any given CFL

- add a transition $((s_0, \varepsilon, \varepsilon), (f, S))$ that starts computation by pushing S , the start non-terminal, onto the stack.
- add transitions for when the top symbol of the stack is a non-terminal. These should pop the non-terminal and push the resulting string. That is,

for every rule $X \rightarrow w$ in the CFG, add a transition $((f, \varepsilon, X), (f, w))$.

PDA for any given CFL

- add transitions for when the top symbol of the stack is a terminal:
 - If this symbol is next on the tape, the transition should read it and also pop the symbol. That is, add transitions $((f, a, a), (f, \varepsilon))$ for every character in the alphabet Σ of the CFG.

PDA for any given CFL

- add transitions for when the top symbol of the stack is a terminal:
 - If this symbol is next on the tape, the transition should read it and also pop the symbol. That is, add transitions $((f, a, a), (f, \varepsilon))$ for every character in the alphabet Σ of the CFG.
- By construction, this PDA will accept any word generated by the CFG. (CFG generated $\Rightarrow$ PDA accepted)

PDA for any given CFL

- add transitions for when the top symbol of the stack is a terminal:
 - If this symbol is next on the tape, the transition should read it and also pop the symbol. That is, add transitions $((f, a, a), (f, \varepsilon))$ for every character in the alphabet Σ of the CFG.
- By construction, this PDA will accept any word generated by the CFG. (CFG generated $\Rightarrow$ PDA accepted)
- To complete the proof, it needs to be argued that the PDA will not recognise any other word apart from the ones generated by the CFG (PDA accepted $\Rightarrow$ CFG generated): the transition rules of the form $((f, a, a), (f, \varepsilon))$ will only be applicable if a previous rule of the form $((f, \varepsilon, X), (f, w))$ has pushed 'a' in the stack as part of w, so only terminals produced by the CFG production rules are processed

PDA pushing symbols one at a time

– We indicated to add:

for every rule $X \rightarrow w$ in the CFG, add a transition $((f, \varepsilon, X), (f, w))$.

If $w = w_1, w_2, \dots, w_n$, this could be equally done adding new states

$q_1, q_2, \dots, q_{n-1}$

And transitions:

$((f, \varepsilon, X), (q_1, w_n))$

$((q_1, \varepsilon, \varepsilon), (q_2, w_{n-1}))$

.

.

.

$((q_{n-1}, \varepsilon, \varepsilon), (f, w_1))$

Note: Since the word is to be read from left to right, the end of the word is pushed first, to be at the bottom of the stack.

Example

- Consider the CFG for Palindrome:

$$S \rightarrow AK \mid BL \mid AA \mid BB \mid a \mid b \mid \varepsilon$$
$$K \rightarrow SA$$
$$L \rightarrow SB$$
$$A \rightarrow a$$
$$B \rightarrow b$$

- Then the corresponding PDA has transitions:

– $((s_0, \varepsilon, \varepsilon), (f, S))$ (for the start transition)

and..

Example (cont.)

..and

- $((f, \varepsilon, S), (f, AK)).$
- $((f, \varepsilon, S), (f, BL)).$
- $((f, \varepsilon, S), (f, AA)).$
- $((f, \varepsilon, S), (f, BB)).$
- $((f, \varepsilon, K), (f, SA)).$
- $((f, \varepsilon, L), (f, SB)).$

$S \rightarrow AK \mid BL \mid AA \mid BB \mid a \mid b \mid \varepsilon$

$K \rightarrow SA$

$L \rightarrow SB$

$A \rightarrow a$

$B \rightarrow b$

- $((f, \varepsilon, S), (f, a)).$
- $((f, \varepsilon, S), (f, b)).$
- $((f, \varepsilon, S), (f, \varepsilon)).$
- $((f, \varepsilon, A), (f, a)).$
- $((f, \varepsilon, B), (f, b)).$

- $((f, a, a), (f, \varepsilon)).$
- $((f, b, b), (f, \varepsilon)).$

Example (cont.)

..and

- $((f, \varepsilon, S), (f, AK)).$
- $((f, \varepsilon, S), (f, BL)).$
- $((f, \varepsilon, S), (f, AA)).$
- $((f, \varepsilon, S), (f, BB)).$
- $((f, \varepsilon, K), (f, SA)).$
- $((f, \varepsilon, L), (f, SB)).$

$S \rightarrow AK \mid BL \mid AA \mid BB \mid a \mid b \mid \varepsilon$

$K \rightarrow SA$

$L \rightarrow SB$

$A \rightarrow a$

$B \rightarrow b$

- $((f, \varepsilon, S), (f, a)).$
- $((f, \varepsilon, S), (f, b)).$
- $((f, \varepsilon, S), (f, \varepsilon)).$
- $((f, \varepsilon, A), (f, a)).$
- $((f, \varepsilon, B), (f, b)).$

- $((f, a, a), (f, \varepsilon)).$
- $((f, b, b), (f, \varepsilon)).$

Example (cont.)

..and

- $((f, \varepsilon, S), (f, AK)).$
- $((f, \varepsilon, S), (f, BL)).$
- $((f, \varepsilon, S), (f, AA)).$
- $((f, \varepsilon, S), (f, BB)).$
- $((f, \varepsilon, K), (f, SA)).$
- $((f, \varepsilon, L), (f, SB)).$

$S \rightarrow AK \mid BL \mid AA \mid BB \mid a \mid b \mid \varepsilon$

$K \rightarrow SA$

$L \rightarrow SB$

$A \rightarrow a$

$B \rightarrow b$

- $((f, \varepsilon, S), (f, a)).$
- $((f, \varepsilon, S), (f, b)).$
- $((f, \varepsilon, S), (f, \varepsilon)).$
- $((f, \varepsilon, A), (f, a)).$
- $((f, \varepsilon, B), (f, b)).$

- $((f, a, a), (f, \varepsilon)).$
- $((f, b, b), (f, \varepsilon)).$

CFL for any given PDA

- **Theorem:** There exists an algorithm that, given any PDA, constructs a CFG such that the language generated by the CFG is exactly that accepted by the PDA.
- Proof: Omitted.

Regular languages

- Note that the class of regular languages is contained in the class of context-free languages.
- That is, every regular language is a context-free language.
- Proof: ?

Regular languages

- Note that the class of regular languages is contained in the class of context-free languages.
- That is, every regular language is a context-free language.
- Proof: An FSA is a PDA which doesn't use its stack. Every regular language has a FSA that recognizes it.

NPDA and DPDA

- Earlier in the course it was demonstrated that for FSA, the deterministic and non-deterministic versions were equivalent.
- Is this also true for PDA?
- The (perhaps surprising) answer to this question is ‘no’.
- That is, NPDA are more powerful (they can recognise more languages) than DPDA.

NPDA and DPDA

- Earlier in the course it was demonstrated that for FSA, the deterministic and non-deterministic versions were equivalent.
- Is this also true for PDA?
- The (perhaps surprising) answer to this question is 'no'.
- That is, NPDA are more powerful (they can recognise more languages) than DPDA.

NPDA and DPDA

- The reason for this is inherent non-determinism in some languages.
- For example, consider a PDA that recognises all strings of form $a^n b^n$.
- This might read an 'a' at a time, pushing a token onto the stack at each point. Followed by a 'b' at a time, popping a token from the stack. This is clearly deterministic.

NPDA and DPDA

- Now consider the language Palindrome.
- Suppose the PDA first sees an 'a' and pushes an 'a' onto the stack. Suppose the next symbol on the tape is also an 'a'.
- Should the PDA pop the stack because it is matching the reflected second half of the palindrome?
- ...or push another 'a' onto the stack because the midpoint of the tape has not yet been reached?

NPDA and DPDA

- If the midpoint is not known, then the machine does not know which option to take.
- Therefore, it must explore all possible options (and a fully mechanised machine needs a rule on how to do this search).

Non-context free languages

- Having seen that there are some languages that are not regular, and some techniques to determine whether or not they are, a similar question arises for CFLs.
- **Are there non-context-free languages?**
- Can it be determined whether or not a language is context-free?

Pumping lemma for CFLs

- **Lemma:** Let G be a context-free grammar. Then there exists a number k such that every string w in the language of G with $|w| \geq k$ can be written in a partition

$$w = uvxyz, \text{ for some strings } u, v, x, y, z$$

such that:

- $|v| > 0$ or $|y| > 0$
- $|vxy| \leq k$
- for any $i \geq 0$, uv^ixy^iz is in the language of G .

Summary

- CFLs can be described by CFGs.
- Pushdown automata accept context-free languages.
- A PDA can be constructed for a given CFG (and vice versa).
- DPDA $\neq$ NPDA

Reading

From Sipser:

- Pages 101-111 for context-free languages and grammars, as well as parse trees.
- PDA, 111-124
- Pumping lemma for CFGs, 125-129